generate_sample_data_notebooks

New notebook

```
In [ ]: df = spark.read.format("csv").option("header","true").load("Files/customer_data.
        # df now is a Spark DataFrame containing CSV data from "Files/customer_data.csv"
        display(df)
```

This is a native Python script that generates data for a subscription service

- Data modelling
- engineering
- performance analytics (Power BI)

This is a utility script that can be customized to regenerate any type of sample data for development, testing or other uses

There are 2 stages in order to write the csv file to the base folder 'Files to maintain the file as an intact csv file (non-partitioned), because by default file is written into DELTA format across multiple nodes - partitions.

```
So if you call e.g.
df_spark.write.mode("overwrite").option("header",
True).csv("Files/customer_data")  #writes it to lakehouse as
partitioned files
```

We use **mssparkutils** from **notebookutils** library for the file movements - Only works in Microsoft Fabric

1. So Stage 1: Write to a temporary location by Coalescing file to a single partition (df_spark.coalesce(1)) value 1 indicating combine all into 1 partition
2. Stage 2: move the file to the base Lakehouse DELTA 'FILES' folder

```
In [ ]: import pandas as pd
        import random
        import numpy as np

        from notebookutils import mssparkutils  # Only works in Microsoft Fabric

        %pip install faker

        from faker import Faker



        fake = Faker()
        Faker.seed(42)
        random.seed(42)
        np.random.seed(42)

        # Constants
        num_rows = 2000
```

```python
subscription_types = ['Free', 'Premium', 'Trial', 'Enterprise', 'Unknown']
country_codes = ['United States', 'United Kingdom', 'Nigeria', 'India', 'Germany

referral_pool = [fake.lexify(text='REF????') for _ in range(100)] + [''] * 100

def generate_customer_id(i):
    return f'C{str(i).zfill(4)}'

def generate_email():
    email = fake.email()
    if random.random() < 0.1:
        email = email.upper()  # Inconsistent casing
    if random.random() < 0.05:
        email = email.replace("@", "")  # Malformed
    return email

def generate_signup_date():
    if random.random() < 0.05:
        return "32-13-2022"  # Impossible date
    elif random.random() < 0.05:
        return "not-a-date"  # Invalid format
    else:
        return fake.date_between(start_date='-5y', end_date='today').strftime('%

def generate_age():
    r = random.random()
    if r < 0.05:
        return None  # Null
    elif r < 0.1:
        return fake.word()  # Non-numeric
    elif r < 0.15:
        return -random.randint(1, 20)  # Negative age
    elif r < 0.2:
        return 0  # Zero age
    else:
        return random.randint(18, 90)

def generate_income():
    if random.random() < 0.1:
        return '$abc'  # Invalid income
    return f"${random.randint(20_000, 120_000):,}.00"

def generate_last_login():
    if random.random() < 0.1:
        return "malformed_timestamp"
    elif random.random() < 0.1:
        return ""  # Missing
    else:
        return fake.date_time_between(start_date='-2y', end_date='now').isoforma

def generate_referral_code():
    return random.choice(referral_pool)

# Generate data
data = {
    "customer_id": [generate_customer_id(i) for i in range(1, 2001)],
    "email": [generate_email() for _ in range(2000)],
    "signup_date": [generate_signup_date() for _ in range(2000)],
    "age": [generate_age() for _ in range(2000)],
    "income": [generate_income() for _ in range(2000)],
    "country_code": [random.choice(country_codes) for _ in range(2000)],
    "phone_number": [fake.phone_number() for _ in range(2000)],
    "subscription_type": [random.choice(subscription_types) for _ in range(2000)
    "last_login": [generate_last_login() for _ in range(2000)],
    "referral_code": [generate_referral_code() for _ in range(2000)],
}

df = pd.DataFrame(data)

# Convert pandas DataFrame to Spark
df_spark = spark.createDataFrame(df)

# Write to Lakehouse in CSV format
#df_spark.write.mode("overwrite").option("header", True).csv("Files/customer_dat

#df_spark.coalesce(1).write.mode("overwrite").option("header", True).csv("Files/

#I want the file saved in the base 'Files' folder as is - needs a few steps urgg
df_spark.coalesce(1) \
    .write.mode("overwrite") \
    .option("header", True) \
    .csv("Files/tmp_customer_data")

'''
# List the files written to the temp folder
#files = dbutils.fs.ls("Files/tmp_customer_data")

# Find the actual CSV file (skip _SUCCESS or metadata files)

#csv_file = [f.path for f in files if f.path.endswith(".csv")][0]

# Move it to the base Files folder with a proper name

#dbutils is a databricks native module
#dbutils.fs.mv(csv_file, "Files/customer_data.csv")

#df.to_csv("customer_data.csv", index=False)

#Optional - delete the temporary temp location
#dbutils.fs.rm("Files/tmp_customer_data", recurse=True)
'''

# List files in the temp folder
files = mssparkutils.fs.ls("Files/tmp_customer_data")    # only a microsoft modu

# Find the part file (the real CSV)
csv_file_path = [f.path for f in files if f.path.endswith(".csv")][0]

# Move and rename to customer_data.csv in the root Files folder
mssparkutils.fs.mv(csv_file_path, "Files/customer_data.csv")

# Optional: clean up temp folder
mssparkutils.fs.rm("Files/tmp_customer_data", recurse=True)


print("customer_data.csv has been created.")
```

```
In [4]: #save to SQL table
        df_spark.write.mode("overwrite").saveAsTable("customer_data")
```

```
StatementMeta(, 5f8e7246-53ac-415d-95e3-6f9defc031c7, 18, Finished, Available, Fi
nished)
```

```
In [5]: df = spark.sql("SELECT * FROM laserengravelakehouse.customer_data LIMIT 1000")
        display(df)
```

```
StatementMeta(, 5f8e7246-53ac-415d-95e3-6f9defc031c7, 19, Finished, Available, Fi
nished)
SynapseWidget(Synapse.DataFrame, 6a79421c-9a2f-48c7-b9ff-86cfe478d39a)
```

```
In [1]: df = spark.read.format("csv").option("header","true").load("Files/customer_data.
        # df now is a Spark DataFrame containing CSV data from "Files/customer_data.csv"
        display(df)
```

```
StatementMeta(, 63cbaa19-ceb2-4fcb-98b0-aef048eaf193, 3, Finished, Available, Fin
ished)
SynapseWidget(Synapse.DataFrame, ef7f8219-00d8-4125-8025-76d8daeb0bba)
```

```
In [ ]: df = spark.sql("SELECT * FROM laserengravelakehouse.customer_data LIMIT 1000")
        display(df)
```